

## Sample Questions for Midterm (CIS 635)

On the actual midterm, you will have plenty of space to put your answers.  
Some of these questions may be reused for the exam.

1. Given the following OCAML code:

```
let x = 3;;
let f y = x + y;;
let x = 5;;
let z = f 2;;
let x = "hi";;
```

What value will **z** have? Will the last declaration (**let x = "hi";**) cause a type error?  
What is the value of **x** after this code has been executed?

2. Given the following OCAML datatype:

```
type int_seq = Null | Snoc of (int_seq * int)
```

write a tail-recursive function in OCAML **all\_pos : int\_seq -> bool** that returns **true** if every integer in the input **int\_seq** to which **all\_pos** is applied is strictly greater than 0 and **false** otherwise. Thus **all\_pos (Snoc(Snoc(Snoc(Null,3),5),7))** should return **true**, but **all\_pos (Snoc(Null,~1))** and **all\_pos (Snoc(Snoc(Null, 3),0))** should both return **false**.

3. Write an OCAML function **pair\_up** takes first a function, then an input list and returns a list of pairs of an element from input list (the second argument), paired with the result of applying the first argument to that element. What is the OCAML type of **pair\_up**? What is the result of the following expressions:

- pair\_up (fun x -> x + 3) [6;4;1];;**
- pair\_up ((fun x -> "Hi, ^x), ["John"; "Mary"; "Dana"]);;**
- pair\_up (fun x -> x \*. 2.0);;**

4. Using the rules provided in class, derive a valid type judgment for

```
let rec fact n = if n = 0 then 1 else let r = fact (n - 1) in n * r;;
```

(The rules will be provided for you on the exam, if this kind of question is asked.)

5. Write the definition of an OCAML variant type **reg\_exp** to express abstract syntax trees for regular expressions over a base character set of booleans. Thus, a boolean is a **reg\_exp**, epsilon is a **reg\_exp**, the concatenation of two **reg\_exp**'s is a **reg\_exp**, the "choice" of two **reg\_exp**'s is a **reg\_exp**, and the Kleene star of a **reg\_exp** is a **reg\_exp**.

6. Give a (most general) unifier for the following unification instance. Capital letters denote variables of unification. Show your work by listing the operation performed in each step of the unification and the result of that step.

$$\{X = f(g(x), W), h(y) = Y, f(Z, x) = f(Y, W)\}$$

7. In the  $\lambda$ -expression below, write under each arrow whether the indicated variable occurrence is free or bound (write F or B). Also, for each bound occurrence, draw an arrow back to the abstraction that binds it.

$$\lambda x. (z (\lambda y. \lambda x. y z x) (y x))$$

$\uparrow \qquad \qquad \qquad \uparrow \uparrow \uparrow \uparrow \uparrow$

8. Evaluate the following  $\lambda$ -expression to  $\alpha\beta$ -normal form, if one exists or explain why one does not exist. Show all work.

$$(\lambda x. \lambda y. \lambda z. y z x) (\lambda x. x x) (\lambda x. \lambda y. y x) (\lambda x. x)$$

9. Evaluate the following  $\lambda$ -expression as far as possible using each of lazy evaluation and eager evaluation. If either evaluation fails to terminate, write “Diverges” and a brief explanation why the evaluation fails to terminate.

$$(\lambda y. \lambda x. y x x) (\lambda x. \lambda y. x x) ((\lambda x. x (\lambda y. y)) (\lambda x. x))$$

10. Represent the constructors for the following OCAML type in the lambda calculus

**type answer = yes | no | maybe**

Write the lambda term that returns the representation of **yes** when applied to the representation of **no**, the representation of **no** when applied to the representation of **yes**, and the representation of **maybe** when applied to the representation of **maybe**.

11. For each of the following languages, give a regular expression describing it:

- The set of all binary strings with at least three ones
- The set of all binary strings with exactly one occurrence of 111
- The set of all binary strings with an even number of zeros.

12. Match each of the following finite state automata with each of the following regular expressions that describes the same language:

- $f(gvf)^*$
- $f^*fg$
- $(gvf)^*f^*fg(gvf)^*$
- $ff^*g(gvf)^*$
- $f^*ff^*(gvf)^*$

